XMI based UML processing in KobrA (Position Paper)

Joanna Filipek and Marko Fabiunke
Institute for Computer Architecture and Software Technology
German National Research Center for Information Technology
Kekuléstraße 7, 12489 Berlin, Germany

Abstract *KobrA is the name of a project concerned with the development of a truly component-based software development method as well as tools and technologies supporting this method. Different to other component technologies, KobrA components cover the complete development life-cycle, hence KobrA components are not only "binary" modules, but come along with a complete set of descriptions notable with their own UML model. Since a software system usually consists of several components, the management of UML models and their inter-relationships, dependencies and constraints plays an important role within the technological part of the KobrA project. This paper gives a glimpse of the work in progress concerning the handling and interpretation of UML models within KobrA.*

1 Introduction

The software industry is currently pinning its hopes for future software productivity, quality and maintainability gains on component-based software engineering (CBSE). However, there is less consensus about what precisely a component is and how it should be applied in practical software development scenarios. One point most contemporary component technologies (e.g. DCOM, CORBA, JavaBeans) agree on is that components are binary, ready-to-run software items that can be deployed immediately without recourse to the edit/compile/link cycle of traditional software technologies. Therefore, the component paradigm has so far only penetrated the "implementation" phase of the software life-cycle, and does not yet play a major role in the earlier analysis and design activities of large software projects. As a result, many of the potential benefits of CBSE are not fully realized in practical software development settings.

The KobrA project addresses this problem by making components the focus of the entire software development life-cycle, not just the implementation and deployment activities, and by adopting a product-line strategy for their creation, maintenance and deployment. The resulting approach augments the typical "binary-module" view of components with a full, UML-based representation of their characteristics and relationships. In KobrA components are described by a combination of UML diagrams and textual documents at a level of abstraction akin to analysis and design in traditional approaches. This allows the essential structure and behavior of component-based systems to be described in a way that is independent of (but compatible with) specific component technologies.

The aim of the KobrA project is the development of a truly component-based software development method covering the complete development life-cycle, as well as techniques and technologies supporting this method, the later eventually resulting in the construction of a workbench enabling users to use the KobrA method by providing method specific tools to manage and guide the KobrA development process as well as the developed products. Since the UML plays an important and significant role in the KobrA method, many of the technological problems deal with UML processing, hence much work in KobrA has been placed around the interpretation, manipulation and transformation of UML models from the KobrA component point of view. This paper focuses on the later questions rather than to give a complete description of the KobrA project. More informations about the KobrA project can be found in [1, 2].

The KobrA project is funded by the German Ministry for Research and Education (BMBF) and is being conducted by four partners, two industrial companies and two research organizations. Those are: Softlab GmbH, Munich (project coordinator), Psipenta GmbH, Berlin, Fraunhofer IESE, Kaiserslautern and GMD FIRST, Berlin.

2 Artifacts

The central artifact in KobrA is the K(obrA)-component (or Komponent). Like other components, a Komponent is a logically self-contained package of functionality, with well-defined interfaces, that can be readily combined with other components to create larger units (components). The main difference between KobrA components and regular components is that they are described by a combination of UML-diagrams and textual documents at a level of abstraction akin to traditional analysis/design. Binary representations of components are still available, but only as one specific incarnation of Komponents.

The description of a Komponent is split into two main parts: the specification, which describes the externally visible characteristics of the Komponent and thus defines the "requirements" which it is expected to meet, and the realization which describes how the Komponent satisfies these requirements in terms of interactions with other Komponents and, therefore, captures the architecture (or design) of the Komponent. The specification of a Komponent describes its externally visible properties primarily in terms of UML class and object diagrams, UML structural diagrams and textual operation specifications. The realization of a Komponent primarily extends these descripions in terms of more detailed UML class and object diagrams, UML interaction diagrams and UML activity diagrams. Well-defined inter-model consistency and realization relationships ensure the quality of the Komponent.

In essence KobrA views (and treats) a Komponent as a system in their own right. If it makes sense, a Komponent can be executed as a system on its own. However, in most cases it will be combined with others to define a larger more powerful system (i.e. Komponent).

KobrA distinguishes between two such systems, KobrA frameworks and KobrA applications. Basically, a framework provides a generic description of the software elements making up a family of applications. In contrast with most other approaches, a KobrA framework embodies all concrete variants of a family, not just the common parts. This is achieved by capturing all possible features within the framework and using decision models to describe the choices that distinguish distinct members (i.e. applications) of the family. A KobrA framework is a tree-structured hierarchy of Komponents, in which the parent/child relationship represents composition (a parent is composed-of its children). The involved Komponent specifications and realizations are interrelated by carefully controlled consistency, trace-ability and realization relationships. Komponents are called to be generic, when they describe all the features of a family of Komponents under the control of a decision model, they are concrete, when only the properties of a member of the family are defined.

3 Method

The KobrA Method defines the products and processes involved in creating and applying a framework of KobrA Komponents. The method is based on the fundamental principle of cleanly separating the product from the process, and also of separating concerns within development and maintenance. The result is a method that is highly-architecture centric, incremental and scalable.

Since the method adopts the product-line philosophy, the first major activity is the creation of a reusable framework which, in KobrA, is a tree-shaped hierarchy of Komponents, organized in terms of the composition relationship. As a product, a KobrA framework is regarded as being in a consistent state when all the consistency, realization and contract relationships have been fully validated and quality controlled. The Framework engineering activity is a highly recursive process, which applies the same basic set of techniques to all Komponents, regardless of their granularity or their position in the composition hierarchy. Once complete, a framework can be instantiated to provide specific applications, tailored to the needs of specific customers. The result is an application with the same form and structure as the framework, but with all genericity and unrequired features removed. The application can then be transformed into an equivalent implementation that contains the source code for automated compilation tools.

4 Workbench

For industrial software development the KobrA method should be supported by a set of specially developed tools and technologies to guide and manage the process of Komponent development. Therefore, the KobrA project also works on a repository based software development platform known as the KobrA workbench. The workbench consists of three main parts, the KobrA Komponent repository, the Komponent manager and the Komponent desk.

The repository is the heart of the workbench and used to store and manage all created artifacts of the KobrA method. It is based on the commercial product "Enabler" developed by Softlab, one of the industrial partners of the KobrA project. Enabler is an open object repository system providing support for integrated storage and management of data, flexible and high-performance access to this data by users and applications, team coordination and version management.

The Kobra manager works on top of the repository and contains all the functionality specific to the KobrA method and not provided already by the repository. This includes all the consistency checking mechanisms required by the KobrA method as well as the configuration techniques coming along with the KobrA decision model.

The KobrA desk is the part of the workbench the user works with. It consists of a set of tools to create, manipulate and visualize the KobrA artifacts. Some of these tools are standard tools like text processing tools, UML design tools or those delivered with the Enabler repository product. Others have been newly developed to help the user to manage and visualize all the KobrA artifacts and there manifold relations.

5 UML processing

As has been mentioned above, each Komponent is treated as a system in its own and comes along with two UML models, one for the Komponent specification and one for the Komponent realization. The realization model can be viewed as an extension of the specification model, since it should contain (import) the entire specification. On the other hand, the realization of a Komponent may be achieved by using other Komponents, hence, the realization model needs to know (import) the specification model of those Komponents (or parts of it). Therefore, all UML models involved in a framework of Komponents are somehow interrelated to each other. A change in the specification of some Komponent has an impact (downwards) to its realization model as well as (upwards) to the realization model of the Komponents using this Komponent.

In theory, KobrA comes along with some well-defined consistency rules that can be evaluated to validate the consistency between the different UML models within a KobrA framework. Some of these rules are specific to the KobrA method, e.g. the specification model should contain only one class having methods (which is the class representing the Komponent). Others simply ensure the consistency of the shared modeling elements, e.g. the imported UML models of some Komponent model should not be changeable within that model, but a change in the original model should be triggered to the importing models. The KobrA method somehow requires a system being able to manage such a set of interrelated UML models. However, there is currently no (standard) UML design tool available, supporting this behavior. Moreover, since KobrA defines some additional constraints with respect to the allowed modeling elements a specification/realization model could contain, the KobrA system needs access to all the details of an UML model, being able to compare this model against others or to check the modeling elements within a model.

A possible but simple solution to some of the consistency problems is, to put all Komponent models into one big model for the whole framework. This has been done as a case study with a simple banking example system. In that study, each Komponent is mapped to a package within that model and a class within the package. The package structure mirrors the parent-child relationship between the Komponents and reflects the fact, that a KobrA Komponent is a class as well as a subsystem within the entire framework. Importing parts of a package to other packages can be considered as equivalent to importing a specification of some Komponent to other Komponents. Navigation through the tree-structure of the framework is therefore the same as navigation through the package structure.

However, the above approach is somehow against the (ideal) KobrA method. KobrA penetrates the user with the principle of locality, meaning, the developer/user should only see what he is really working on. Hence, someone working on the specification or realization of some Komponent should not need to see the other parts of the framework, except those imported from other Komponent models. Moreover, this approach does not really separates the specification from the realization of a Komponent. It always contains the complete (realization) model. Visibility restrictions can only be obtained by using the model/view concepts of the underlying UML design tool, but have to be created manually by the user. KobrA specific constraints and consistency rules can also only be validated by the user, since the UML design tool has no idea about the user developing a KobrA Komponent.

Although the above "workaround" can be used for smaller projects, a more KobrA-like solution is needed for the workbench. This solution has been placed around the usage of the XML-based Metadata Interchange standard (XMI). XMI has been proposed by the OMG (which is also responsible for the UML standard) as a standard method for text-based exchange of object-based data on the base of XML. It is not the only one of XML-based exchange formats, but the one that is supported by the developers of UML. Moreover, many UML tool providers have proclaimed to support the exchange of UML models via XMI in future releases of their products. Rational Rose provides a free plug-in for their UML case tool, allowing the import and export of XMI-based UML models. TogetherSoft has a build-in support for XMI and the XMI toolkit (from IBM) supports the transformation of Rational Rose model files to Java source code via the XMI standard. Other firms have proposed interest in the XMI standard (e.g. Select Enterprise), but have not been coming up with solutions so far.

In the KobrA workbench scenario, any UML design tool can be used to create and manipulate the UML models for the KobrA Komponents, since the KobrA repository allows to store (and manage) any kind of file. However, additional support for the KobrA method (e.g. automatic consistency checks) can only be incorporated, when the tool is able to (additionally) save their models on the base of the XMI standard. At present, only the UML 1.1 (and XMI 1.0) standard are supported, but this will change later on this year to meet the newer standards UML 1.3 and XMI 1.1. An XMI loader (being part of the KobrA manager) is able to parse the exported XMI files and creates a detailed representation of the contained model in the KobrA repository. This representation consists of objects according to the UML meta-model as specified by the UML standard. Starting from a root object (representing the model itself) any detail of the so stored model can be obtained by traversing the object attributes as well as the links to other objects.

The so stored models can be used later on to apply any of the consistency checks defined by the KobrA method. Within the current development phase, such checks have to be implemented (in Java) by hand, but plans are made to support the OCL language and the execution of any valid OCL script within the KobrA workbench at a later stage of development. At present, a Java API for UML models based on the UML standard is under development. This API should support a standard way to traverse an UML model based on the UML meta-model description. The standard implementation of that API will work an the stored UML models in the repository, but implementations based on in-memory representations or accessing a Rational Rose model via the COM API, provided by the Rational Rose tool, should be possible as well.

One of the interesting point of being able to traverse a UML model within the KobrA repository is the fact, that this helps to develop the KobrA workbench system itself. Since the UML meta-model can be described as a UML model, we stored the UML meta-model in our repository first. Traversing this model allows now the generation of parts of the functionality of the workbench. The above mentioned UML model API will be generated completely by such a generation tool currently under development. Moreover, the Enabler data model (used to store KobrA artifacts as well as UML models) will be generated in the next version out of an XMI file containing the KobrA meta-model (which is indeed only an extended UML meta-model). Those, we should be able to adopt the current data model (based on UML 1.1) to new standards (e.g. UML 1.3) without any reprogramming.

6 Summary

The KobrA approach to component-based software development is based on the idea of making components to be the focus of the entire software development life-cycle. KobrA Komponents are therefore not only binary products, but well-designed systems by there own. Each Komponent comes along with a complete UML model as well as other descriptive documents.

In KobrA a system (framework/application) comes along as a tree of KobrA Komponents with contract and usage relationships defined between the individual Komponents. Hence, a KobrA system can be viewed as a system of interrelated UML models. Since none of the currently available UML tools supports that kind of interrelated UML model management, it has been left to the KobrA workbench to provide a solution to that problem. The KobrA workbench addresses this problem by making the XMI exchange standard to be the key technology of processing UML models (from different sources) within the KobrA workbench. This allows the user to store and analyze UML models in detail, such that several processing scenarios like inter-model consistency checks, KobrA-specific model constraint validation as well as the transformation of UML models into other representations (code, data models) are available. Due to the growing interests by our partners, the technology will be adopted to other ongoing projects beyond KobrA as well.

References

[1] C. Atkinson, J. Bayer, and D. Muthig. Component-based product line development. the kobra approach. In *Proc. of the 1th Software Product Lines Conference (SPLC1)*, pages 289–309, 2000.

[2] D. Muthig and J. Bayer. Helping small and medium-sized enterprises in moving towards software product lines. In *Proc. of the Workshop Software Product Lines. Economics, Architectures, and Implications.*, pages 98–101, Limerick, 2000.